グリッド上での並列分枝限定法アプリケーション

合田 憲人[†‡]　　大角 知孝[†]

本稿では，グリッド上で分枝限定法アプリケーションを効率よく実行するための並列化手法とその性能評価結果について報告する．本アプリケーションは，グリッド上で細粒度タスクを効率よく実行するために，階層的マスタ・ワーカ方式により並列化され，複数の WAN 上の PC クラスタ群から構成されるグリッド上に GridRPC ミドルウェアである Ninf-G および Ninf を用いて実装される．本アプリケーションをグリッド実験環境上に実装し，性能評価を行った結果，Ninf-G および Ninf を組み合わせて実装された階層的マスタ・ワーカ方式は，実行時間が 1[sec] 未満の細粒度タスクから構成されるアプリケーションをグリッド上で効率よく実行可能であることが確認された．

Parallel Branch and Bound Application on the Grid

Kento Aida[†‡], Tomotaka Osumi[†]

This paper presents a case study to effectively run the parallel branch and bound application on the Grid. The application discussed in this paper is a fine-grain application and is parallelized with the hierarchical master-worker paradigm. The application is implemented on the Grid, which is constructed by multiple PC cluster connected to WAN, by using GridRPC middleware, Ninf-G and Ninf. The experimental results on a Grid testbed showed that implementation of the application with the hierarchical master-worker paradigm using combination of Ninf-G and Ninf effectively utilized computing resources on the Grid testbed in order to run the fine-grain application, where average computation time of the single task was less than 1[sec].

1. はじめに

グリッド計算は，地理的に分散した計算資源を利用することにより，低コストで高性能計算を実現する新しい計算手法として注目されており，従来の大規模計算に要していた時間を短縮するだけでなく，高性能計算の応用範囲を広げる効果も期待されている．しかしグリッド計算環境では，WAN 上の通信オーバーヘッドやグリッド上のセキュリティ処理によるオーバーヘッドが大きいため，効率よく実行可能なアプリケーションは，これらオーバーヘッドに対して十分な計算時間（例えば数十秒～数百秒）を持つタスクから構成されるものに限られている[1][2][3][4]．しかしながら，計算時間の小さなタスクを大量に処理する必要がある大規模細粒度アプリケーションも存在し，これらのアプリケーションをグリッド上で実行することは困難な状況にある．従って，細粒度アプリケーションをグリッド上で効率よく実行させる手法を開発することは，グリッド計算の応用範囲を広げる意味で重要である．

本稿では，グリッド上で分枝限定法アプリケーションを効率よく実行するための並列化手法とその性能評価結果について報告する．分枝限定法は最適化問題解法の一つであり，オペレーションズリサーチ，制御工学，マルチプロセッサスケジューリング等の多くの工学分野で用いられている[5][6][7]．これらのアプリケーションの多くは，細粒度アプリケーションであり，実行時間の短いタスクを大量に処理する必要がある．本稿が述べる並列分枝限定法アプリケーションは，細粒度タスクをグリッド上で効率よく実行するために，階層的マスタ・ワーカ方式を用いて並列化される[8]．階層的マスタ・ワーカ方式では，WAN 上の複数の PC クラスタ間，および各 PC クラスタ内の計算ノード間の 2 階層において，マスタ・ワーカ方式を用いたタスク割り当てが実行される．本方式では，マスタと複数ワーカから構成されるプロセスグループを同一の PC クラスタ上で実行させることにより，頻繁に発生する

† 東京工業大学／Tokyo Institute of Technology
‡ 科学技術振興機構さきがけ／PRESTO, JST

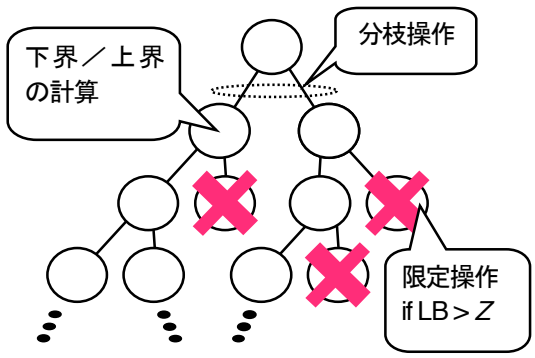

図 0. 探索木

マスタとワーカ間の通信が PC クラスタ上の高速ネットワーク内に局所化されるため，通信オーバーヘッドを軽減することができる．また，各 PC クラスタ上での未処理タスク数に応じて，WAN を介したタスクの移動が行われ，PC クラスタ間の適切な負荷分散が実現される．

本アプリケーションは，GridRPC ミドルウェアである Ninf-G[9]および Ninf[10]によりグリッド上に実装される．GridRPC は，グリッド上でクライアント・サーバ型の遠隔手続き呼び出しを実現するためのプログラミングモデルである．本実装では，WAN にまたがる複数の PC クラスタ間の通信は Ninf-G によりセキュリティ上安全に実行され，PC クラスタ内の通信は Ninf により高速に実行される．

本アプリケーションをグリッド実験環境上に実装し，性能評価を行った結果，Ninf-G および Ninf を組み合わせて実装された階層的マスタ・ワーカ方式は，実行時間が 1[sec]未満の細粒度タスクから構成される並列分枝限定法アプリケーションをグリッド上で効率よく実行可能であることが確認された．

以後，2 節では本稿が対象とする並列分枝限定法アプリケーションとグリッド上での並列化手法について述べ，3 節ではアプリケーションのグリッド上への実装について述べる．次に 4 節ではグリッド実験環境上でのアプリケーションの性能評価結果を示し，5 節では関連研究について述べる．最後に 6 節では本稿のまとめと今後の課題を述べる．

2. 並列分枝限定法アプリケーション

本節では，分枝限定法について概説するとともに，階層的マスタ・ワーカ方式を用いた並列化手法について述べる．

2.1 分枝限定法

分枝限定法は，与えられた問題を複数の子問題に分割することを再帰的に繰り返すことにより，最適解を探索する手法である．生成された子問題では，目的関数の下界，上界および解（暫定解）が計算される．また，下界および上界の値は，冗長な解探索を省く限定操作および計算の終了判定にも用いられる．

分枝限定法の手順は，図 0に示される探索木によって表される．分枝限定法では，初めに探索木の根ノードに相当する問題を複数の子問題（図中では 2 個の子ノードに相当）に分割する．本操作は，**分枝操作**と呼ばれる．次に生成された各子問題について，目的関数の下界と上界が計算され，さらに，これまでに計算された子問題間で上界を比較することにより，その最小値である**暫定値**を計算する．（本稿では目的関数を最小化する最適化問題を扱っている．）以上の操作は，生成される子問題について再帰的に繰り返され，図 0 に示される探索木が生成される．暫定値は，冗長な探索を省くために行われる**限定操作**に用いられる．限定操作では，各子問題についてその下界（LB）と暫定値（Z）との比較が行われ，下界が暫定値よりも大きい子問題については，それ以上分枝操作を続けても最適解が得られないため，探索木から削除する．最後に，以上の操作を繰り返すことにより得られる暫定値と最小下界が一致するかその差が一定の閾値以下になった場合，全体の計算が終了する．

2.2 階層的マスタ・ワーカ方式による並列化

分枝限定法では，異なる子問題の計算は独立であるため，マスタ・ワーカ方式によりこれら子問題を並列に計算することができる[1][3][11]．本方式では，マスタと呼ばれる 1 プロセスが，探索木上の未処理子問題の計算をタスクとして複数のワーカと呼ばれるプロセスに割り当てることにより，子問題計算が並列に実行される．しかし一般に通信オーバーヘッドの大きいグリッド上では，単純なマスタ・ワーカ方式による並列化では，マスタとワーカ間の通信時間が大きくなり，特に細粒度アプリケーションについては十分な実行時間短縮を行えないという問題がある[8]．

階層的マスタ・ワーカ方式は，グリッド上でのマスタ・ワーカ方式の性能低下を防ぐ手法の一つである．本方式では，単一のマスタと複数のワーカからなるグループが複数構成され，スーパバイザと呼ばれるプロセスがこれら複数のグループを統括する．タスクのワーカへの割り当ては，スーパバイザからマスタへの割り当て，マスタからワーカへの割り当てという 2 段階で行われ，ワーカ上の計算結果の回収は，逆の順序で同様に行われる．本方式では，同一グループに属するマスタとワーカを PC クラスタ等の単一計算システム上に配置することにより，マスタとワーカ間の通信を PC クラスタ上の高速ネットワーク内に局所化し，マスタ・ワーカ間の通信オーバーヘッドを軽減できると

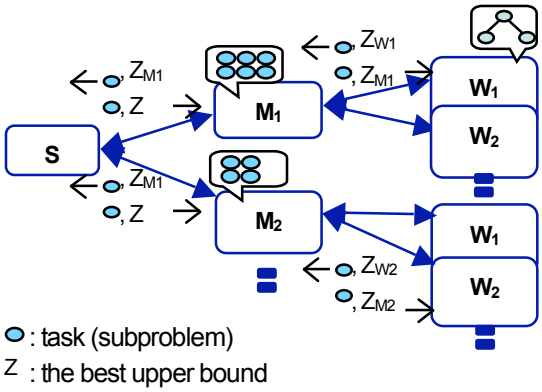

図 0. 階層的マスタ・ワーカ方式による
並列分枝限定法

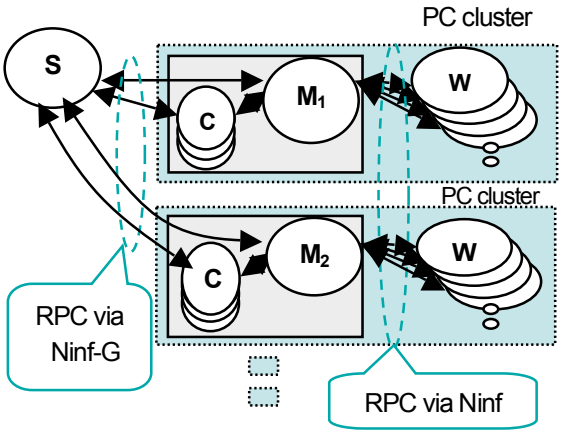

図 0. プロセス配置

ともに, マスタの処理を複数の計算機に分散するため, 単一マスタの処理がボトルネックとなる問題も解決できる. なお, 割り当てを３段階以上で行うアルゴリズムも考えられるが, 本稿では, グリッド環境として複数の PC クラスタが WAN を介して接続されている環境を想定しているため, PC クラスタ間と PC クラスタ内の２段階の割り当てを行うことを想定している.

図 0は, 階層的マスタ・ワーカ方式による並列分枝限定法アルゴリズムの概略を示す§. 図中, **S**, **M**, **W** はそれぞれ, スーパバイザ, マスタ, ワーカを意味し, Z_{Wi}, Z_{Mj}, Z は, それぞれワーカ i 上で計算された暫定値, マスタ j 上に保存されている暫定値, スーパバイザ上に保存されている暫定値をそれぞれ示す.

単一マスタと複数のワーカから構成される各々のグループでは, マスタが複数のワーカに対して探索木上の未処理子問題の処理をタスクとして割り当てる. ワーカでは, 割り当てられた子問題に分枝操作を行って新たな探索木（**部分木**）を生成し, 部分木上の子問題について下界／上界計算, 限定操作を実施した後, 得られた解（暫定解）, 暫定値, 限定操作により削除されなかった子問題をマスタに送信する. マスタは, 受信した子問題を未処理子問題としてマスタ上のキューに登録するとともに, ワーカから受信した暫定値（Z_{Wi}）がマスタ上の暫定値（Z_{Mj}）よりも小さい場合は Z_{Mj} の値を Z_{Wi} の値に更新する. 次にマスタは, アイドル状態のワーカに対して, キュー上の新たな子問題を割り当てると同時に, 更新された Z_{Mj} を送信する. 各ワーカ上で実施される限定操作は, ワーカ上に保存されている暫定値を用いて行われるため, あるワーカ上で算出されたより良い暫定値を他のワーカに迅速に通知することにより, 限定操作の効率が向上し, 全体の計算時間を短縮することができる. またマスタは, ワーカ上で分枝操作により生成される部分木の大きさを指定することにより, ワーカに割り当てるタスクの粒度を調整することができる. これは, ワーカ上の 1 タスク当りの計算量は, ワーカ上の分枝操作により生成される部分木の大きさに依存するためである. ここで, 部分木の大きさは, 部分木の深さにより指定される. 例えば図 0では, 右上のワーカ（W_1）上で深さ 1 の部分木が生成されている.

スーパバイザは, 定期的に各マスタ上の負荷（具体的には未処理子問題数）およびマスタ上に保存されている暫定値（Z_{Mj}）を問い合わせる. ここで, 複数のマスタ間で負荷の不均衡が生じている場合は, スーパバイザは, 高負荷のマスタから低負荷のマスタへ子問題を移動することにより, 負荷分散を行う**. マスタから通知された暫定値（Z_{M_j}）がスーパバイザ上に保存されている暫定値（Z ）よりも小さい場合, スーパバイザは, Z の値を Z_{Mj} の値に更新するとともに, 他のマスタに対して更新された Z を広報する. このように複数のマスタ間で最新の暫定値を共有することにより, ワーカ上での限定操作を促進し, 冗長な探索を防ぐことができる.

3. 実装

本稿では, 異なるサイト, ドメインに属する複数の PC クラスタ群から構成されるグリッド上でアプリケーションを実行することを想定している. このようなグリッド上では, 前節で説明した各プロセスを計算資源へ適切に配置するとともに WAN および LAN 上の効率のよい通信を実現することが重要となる.

3.1 プロセス配置

図 0は, 階層的マスタ・ワーカ方式におけるプロセスの計算資源への配置を表している. 本図では, 複数

§ アルゴリズムの詳細については, 文献[8]を参照されたい.

**負荷分散アルゴリズムの詳細については [12]を参照されたい.

の PC クラスタ（点線の長方形により表示）が WAN を経由して接続されており，図中の S，M，W はそれぞれ，スーパバイザ，マスタ，ワーカを示している．図中 C は，マスタと同一の計算ノード（実線の長方形により表示）上で実行されるプロセスであり，スーパバイザからマスタに対する処理，具体的にはマスタ上の負荷や暫定値の問い合わせ，子問題の移動，暫定値通知等の処理を中継する．

本アプリケーションでは，マスタとワーカ間の通信量がスーパバイザとマスタ間の通信量に比べて非常に大きい傾向がある．そのため，本実装では，単一のマスタと複数ワーカから構成されるプロセスグループを同一の PC クラスタ内に配置し，マスタとワーカ間の通信を PC クラスタ内の高速ネットワーク内に局所化することにより，通信オーバーヘッドを軽減する．

3.2 プロセス間の通信

スーパバイザとマスタ間の通信は，WAN を経由して行われるため，ユーザ認証等のセキュリティを考慮する必要があるが，マスタとワーカ間の通信は同一の PC クラスタ内ネットワークを経由して行われるため，高いセキュリティよりも高速な通信が要求される．そのため本アプリケーションでは，スーパバイザからマスタへの問い合わせや子問題の移動は，Ninf-G によりユーザ認証等を含むセキュリティを考慮して実行される．また，マスタからワーカへの子問題の割り当ては，セキュリティに関する機能を持たないが Ninf-G に比べて高速通信が可能な Ninf により実行される．

Ninf-G[9]は，Global Grid Forum において現在標準化が進められている GridRPC API[13]のリファレンスインプリメンテーションである．Ninf-G により実装されるプログラムは，クライアントプログラムとサーバプログラムから構成され，クライアントプログラムは遠隔計算機上のサーバプログラム（Ninf-G executable）を Ninf-G Client API を通して起動する．クライアントプログラムの開発者は，本 API に従った関数呼び出しコードをプログラム中に挿入することにより，容易にグリッドアプリケーションプログラムを作成することができる．Ninf-G は，グリッド環境を構築するためのソフトウェアツールキットである Globus Tool Kit[14]上に実装されており，クライアントプログラムとサーバプログラム間の通信は，Globus Tool Kit が提供するセキュリティ機能（GSI）を用いて安全に実現される．

Ninf[10]は，Ninf-G の前身として開発された GridRPC ミドルウェアであり，Ninf-G とほぼ同様の API を提供する．Ninf は単体で独立したソフトウェアシステムであり，GSI のようなグリッド上でのセキュリティ機能を提供していないが，Ninf-G に比べて高速な遠隔サーバプログラムの起動が可能である．

3.3 GridRPC による実装

本アプリケーションでは，実行開始時にまずスーパバイザプロセスが起動され，次にスーパバイザが，グリッド上の各 PC クラスタ内の 1 ノードに対して，Ninf-G によりマスタプロセスを起動する．スーパバイザがマスタを起動するための Ninf-G API の例を以下に示す．

```
for(i = 0; i < nMaster; i++){
grpc_function_handle_init(&ex[i],…,"Master");
}

for(i = 0; i < nMaster; i++){
pid[i] = grpc_call_async(&ex[i],…);
}
```

ここで，*nMaster* はマスタプロセス数（＝アプリケーションを実行する PC クラスタ数）を意味する．grpc_function_handle_init() は，遠隔計算機上の Ninf-G executable を呼び出すための関数ハンドルを初期化する API であり，引数として遠隔計算機のホスト名，ポート番号，Ninf-G executable のパス名，計算の入力データが指定される．grpc_call_async()は，遠隔計算機上の Ninf-G executable の非同期呼び出しのための API であり，引数として与えられる関数ハンドルで指定された Ninf-G executable を起動する．

各 PC クラスタ内では，スーパバイザに起動されたマスタプロセスが，Ninf を用いて同一 PC クラスタ内の各計算ノード上にワーカプロセスを起動し，各ワーカに対してタスクを割り当てる．マスタがワーカにタスクを割り当てる際の Ninf API の例を以下に示す．

```
for(i = 0; i < nWorker; i++){
sprintf(ninfURL[i], NINF_URL_LENGTH,
"ninf://%s/Worker", workerList[i]);
exs[i]= Ninf_get_executable(ninfURL[i]);
}

while (1) {
id = Ninf_wait_any();
for (i = 0; i < nWorker; i++)
if (ids[i] == id) break;
            :
ids[i]=Ninf_call_executable_async(exs[i],…);
}
```

ここで *nWorker* は，PC クラスタ内のワーカ数を意味する．Ninf_get_executable()は，サーバプログラム（Ninf executable）を呼び出すための関数ハンドルを

表 1. グリッド実験環境

	specification for a single node	Grid software	RTT [ms]
Client PC	PIII 1.0GHz, 256MB mem. 100BASE-T NIC	GTK 2.2 Ninf-G 1.1.1	
Blade	PIII 1.4GHz x2 512MB mem. 100BASE-T NIC	GTK 2.2 Ninf-G 1.1.1	0.04
Presto III	Athlon 1.6GHz x2, 768MB mem. 100BASE-T NIC	GTK 2.4 Ninf-G 1.1.1	1
Mp	Athlon 1.6GHz x2 512MB mem. 100BASE-T NIC	GTK 2.4 Ninf-G 1.1.1	20
Sdpa	Athlon 2GHz x2, 1024MB mem. 1000BASE-T NIC	GTK 2.4 Ninf-G 1.1.1	14

初期化する API であり，grpc_function_handle_init()と同様の引数が指定される．Ninf_wait_any()は，実行中の Ninf executable の任意の 1 つの終了を待つための API である．Ninf_call_executable_async()は，Ninf executable の非同期呼び出しのための API であり，マスタがワーカにタスクを割り当てるために実行される．

PC クラスタ間の負荷分散および暫定値の通知は，スーパバイザがこれらの処理を実行する Ninf-G executable を起動することにより実現される．例えば，スーパバイザは，各 PC クラスタ内のマスタプロセスが実行されている計算ノードに対して，Ninf-G executable を起動することによりマスタの状態を問い合わせる．起動された Ninf-G executable（図 0中のプロセス C に相当する）は，マスタプロセスから未処理タスク数，暫定値等をプロセス間通信により取得し，これらの情報をスーパバイザに通知する．スーパバイザによるタスクの移動や暫定値通知についても，同様に実現される．

4. 性能評価

本稿の評価に用いられたグリッド実験環境は，表 1 に示す地理的に 4 つのサイトに分散したクライアント PC と 4 台の PC クラスタから構成される．このうち，クライアント PC と表中の Blade は，同一サイト（東京工業大学，神奈川県横浜市）に設置されている．クライアント PC と他の PC クラスタ，具体的には PrestoIII（東京工業大学，東京都目黒区），Mp（徳島大学，徳島県徳島市），Sdpa（東京電機大学，埼玉県比企郡）間の距離はそれぞれ，30[km]，500[km]，50[km]である．表中の RTT は，ping コマンドにより測定したラウンドトリップタイムを示す．また，東京工業大学，東京電機大学，徳島大学間のネットワークは，それぞれ SINET を経由して接続されている．スーパバイザプロセスはクライアント PC 上で実行され，各 PC クラスタ上では，マスタと複数のワーカからなるグループがそれぞれ実行される．また，ユーザおよびホストの認証に用いる証明書は，AIST GTRC CA[15]から発行されたものを用いる．

本評価のベンチマーク問題としては，BMI 固有値問題[5]を用いる．BMI 固有値問題は，以下の式に示す双線形行列関数の最大固有値を最小化する問題であり，制御工学やオペレーションズリサーチの分野で取り扱われている[5][6]．

$$F(x,y) = F_{00} + \sum_{i=1}^{n_x} x_i F_{i0} + \sum_{j=1}^{n_y} y_j F_{0j} + \sum_{i=1}^{n_x}\sum_{j=1}^{n_y} x_i y_j F_{ij}$$

$$F_{ij} = F_{ij}^T,\ i = 0,\ldots,n_x,\ j = 0,\ldots,n_y$$
$$x = (x_1,\ldots,x_{n_x})^T,\ y = (y_1,\ldots,y_{n_y})^T$$

グリッド上で本ベンチマーク問題を求解する場合に発生する通信量は，上式中の係数行列 F_{ij} の大きさ（$m \times m$）および解ベクトルの大きさ（n_x，n_y）より決定される．例えば，マスタがワーカにタスクを割り当てる際に発生する通信量は，以下の式で表される．ただし，D_{in}，D_{out} はマスタからワーカへ通信量，ワーカからマスタへの通信量をそれぞれ意味し，N はマスタとワーカ間で転送される子問題数を示す．

$$D_{in} = 4(m^2 + m) \times (n_x + n_y + n_x n_y + 1) + 28n_x + 28n_y + 28n_x n_y + 136 \text{ [B]}$$
$$D_{out} = N \times (24n_x + 24n_y + 16) + 8n_x + 8n_y + 16 \text{ [B]}$$

一方，スーパバイザがマスタ間の負荷分散を行う場合の通信量は，以下の式で表される．ここで D_{in}，D_{out} はスーパバイザからマスタへ通信量，マスタからスーパバイザへの通信量をそれぞれ意味し，N はスーパバイザによって移動される子問題数を示す．

$$D_{in} = 12 \text{ [B]}$$
$$D_{out} = N * (24n_x + 24n_y + 16) + 4 \text{ [B]}$$

例えば，問題サイズが $n_x = 6$, $n_y = 6$, $m = 24$ である問題を求解する場合，マスタからワーカへのタスク割り当て時に発生する通信量は120[KB]であることに対し，スーパバイザ・マスタ間の通信量は数 B から数 KB 程度であり，WAN 上で発生するスーパバイザ・マスタ間の通信量が PC クラスタ内で発生するマスタ・ワーカ間の通信量に比べて非常に小さいことがわかる．

本アプリケーションの性能は，マスタ間，即ち PC クラスタ間の負荷分散性能に影響を受けると考えられる．本性能評価では，以下に示す負荷分散アルゴリズ

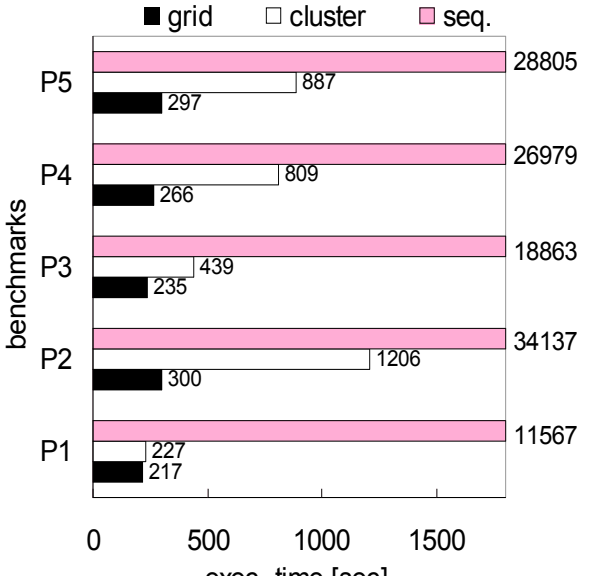

図 0. グリッド実験環境上での実行時間

ムをスーパバイザ上に実装している．本アルゴリズムでは，スーパバイザが各マスタへの問い合わせによりアイドル状態の PC クラスタを発見した場合，各 PC クラスタの性能に応じてタスクが再分散されるように，マスタ間でタスクを移動する．具体的には，スーパバイザはマスタ i 上の未処理タスク数が以下の $N_{task}(i)$ となるようにタスクを移動する．

$$N_{task}(i) = N_{task}(total) \times P(i)$$
$$P(i)=T_{task}(i)\times N_{workers}(i)/\sum(T_{task}(j) \times N_{workers}(j)) ,$$

ここで $T_{task}(i)$ は，PC クラスタ i 上での 1 タスク当りの平均計算時間を意味し，アプリケーション実行中にマスタが記録する実行履歴情報から算出される．また $N_{workers}(i)$ は，PC クラスタ i 上のワーカ数を意味する．

4.1 グリッド実験環境上での評価

図 0は，グリッド実験環境上で 5 種類のベンチマークプログラム（**P1**-**P5**）を求解した場合の実行時間を示す．本実験では，4 サイト上の PC クラスタから合計 348CPU（Blade 上の 73CPU，PrestoIII 上の 97CPU，Sdpa 上の 81CPU，Mp 上の 97CPU）を用いてアプリケーションを実行した．使用したベンチマーク問題では，それぞれ問題サイズは同じ（$n_x = 6$, $n_y = 6$, $m = 24$）であるが，初期値として与えられる係数行列がそれぞれ異なる．図中，**seq** は，Blade 上の 1 ノード上での逐次実行時間，**cluster** は， Ninf を用いたマスタ・ワーカ方式により実装されたアプリケーションによる単一 PC クラスタ（Blade）のみの上での実行時間，**grid** は Ninf-G および Ninf を用いた階層的マスタ・ワーカ方式により実装されたアプリケーションのグリッド実験環境上での実行時間をそれぞれ示す．

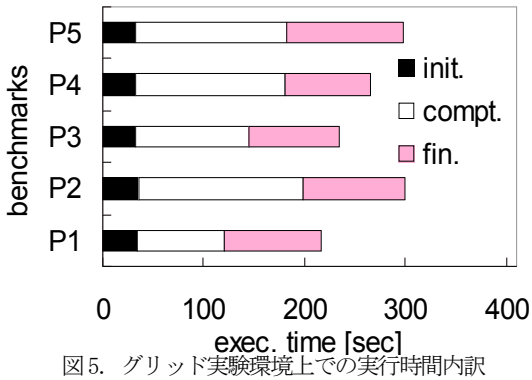

図 5. グリッド実験環境上での実行時間内訳

図中の結果より，マスタ・ワーカ方式により並列化されたアプリケーションの PC クラスタ上での実行時間が逐次実行時間に比べて大きく短縮できていることがわかる．また，グリッド上で階層的マスタ・ワーカ方式を用いてアプリケーションを並列化することにより，実行時間をさらに短縮できることがわかる．例えば最も高い性能を示した **P2** では，逐次実行では，約 9 時間半の実行時間を要していたが，グリッド計算により実行時間が 5 分に短縮されている．また，同様 **P2** のグリッド上での実行時間は，単一 PC クラスタ上での実行に比べても十分に実行時間を短縮できていることがわかる．

図 5 は，ベンチマーク問題のグリッド上での実行時間（**grid**）の内訳を示す．図中，**init**，**compt**，**fin** は，それぞれ Ninf-G プロセスの初期化オーバーヘッド，計算時間，Ninf-G プロセスの終了処理オーバーヘッドをそれぞれ示す．本図の結果より，特に Ninf-G プロセス終了処理に要するオーバーヘッドが大きいことがわかり，これが **P1** に関してグリッド上での性能向上が単一 PC クラスタに比べて小さいこと要因の一つと考えられる．しかしながら，本アプリケーションでは，Ninf-G プロセス終了処理開始前にユーザに計算結果が提供される仕様になっており，ユーザの視点では，図 0に示された実行時間よりも短い時間で計算結果を取得できている．例えば **P1** の場合，ユーザが計算を開始してから計算結果を得るまでの時間は 120[sec]であり，単一 PC クラスタ上での実行に比べて大きく短縮されている．

本評価で用いたベンチマーク問題の 1 タスク当りの平均実行時間は 1[sec]未満である．このような細粒度アプリケーションをグリッド上で従来のマスタ・ワーカ方式を用いて実行した場合，WAN 上で細粒度タスクを転送するオーバーヘッドが大きいため，性能が大きく低下する[8]．これに対して，本評価の結果より，Ninf-G および Ninf を組み合わせて実装された階層的

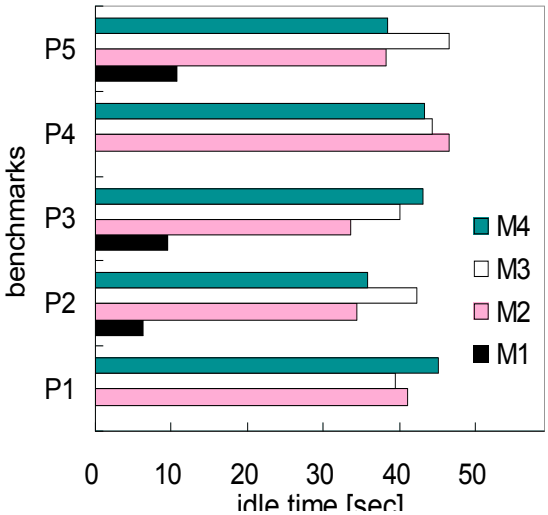

図6. PC クラスタ上でのアイドル時間

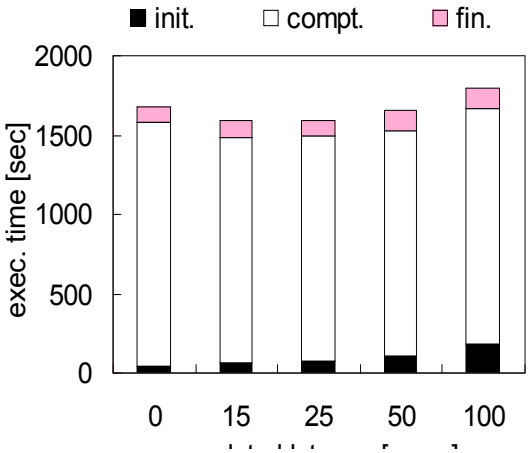

図7. 擬似グリッド実験環境上での実行時間

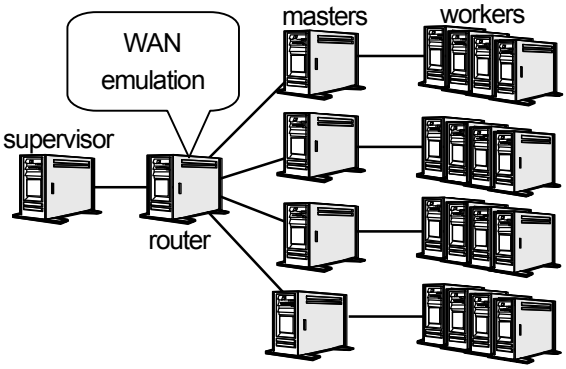

図8. 擬似グリッド実験環境

マスタ・ワーカ方式では，実行時間が 1[sec]未満の細粒度タスクから構成される並列分枝限定法アプリケーションをグリッド上で効率よく実行可能であることが確認された.

4.2 負荷分散

図 6 は，PC クラスタ間の負荷分散性能を評価するために測定した各 PC クラスタ上でのアイドル時間を示す．図中，M1，M2，M3 および M4 は，Blade，PrestoIII，Sdpa，および Mp 内のマスタ，即ち PC クラスタ上のアイドル時間をそれぞれ示す. 本図より，Blade を除いた PC クラスタ上では，30[sec]から 45[sec]程度のアイドル時間が観測されていることがわかる．本アイドル時間の要因を調査するために別途評価実験を行ったところ，これらアイドル時間のほとんどは，アプリケーションの実行開始時にタスク間の並列性が小さいことに起因することが確認された．本アプリケーション中のタスクは，アプリケーション実行開始後に親問題を分割することにより動的に生成されるが，実行開始後しばらくは，グリッド上の全ての計算ノード数に比べて十分なタスクが生成されないため，アプリケーションは単一の PC クラスタ上で実行される．従って，十分なタスクが生成された後は，各 PC クラスタ上のアイドル時間は小さく，PC クラスタ間の負荷分散が効率良く実現されていることが確認された.

4.3 擬似グリッド実験環境上での評価

本アプリケーションの性能は，スーパバイザとマスタ間の通信性能に影響を受けることが考えられる．図 7 は，図 8 に示す擬似グリッド実験環境上での P2 の実行時間を示す．本擬似グリッド実験環境は，LAN 上の複数の PC および PC クラスタを PC ルータにより接続した構成であり，PC ルータ上で NIST Net[16] を用いた WAN 上の通信遅延をエミュレーションすることにより，WAN を介したスーパバイザとマスタ間の通信を擬似的に再現している．図中，**workers** と示された PC は Blade の一部（32CPU）を用い，他の PC は P4 2.4GHz, 1GB mem., 1000BASE-T NIC から構成されている．本評価では，スーパバイザとマスタ間の通信遅延が異なる条件（0[msec] ～ 100[msec]）下での実行時間を測定している．例えば通信遅延が 100[msec]の条件は，日米間ネットワークの通信遅延を想定している.

図 7 より，スーパバイザとマスタ間の通信遅延が 100[msec]といった大きな条件においても，本アプリケーションの性能低下は小さく抑えられていることがわかる．性能低下の主な要因は，初期化処理オーバーヘッドの増加によるものである．以上の結果より，GridRPC を用いた階層的マスタ・ワーカ方式により並列化されたアプリケーションは，通信遅延の大きな WAN 上においても，効率よく実行されることが確認された.

4.4 ユーザインターフェース

本アプリケーションでは，図 9 示すような Web を経由してグリッド上でのアプリケーション実行を可能とするユーザインターフェースを提供している．ユーザは，本インタフェースを通して，入力パラメータのアップロード，計算の実行開始および停止，途中および最終結果の観測を行うことができる．図中，上段のウィンドウは目的関数の下界と上界の途中計算結果の推移，下段のウィンドウは未処理のタスク数を表示している．これらの途中結果は，本アプリケーションのユーザにとって有益な情報であり，ユーザは，途中結果の推移によっては，計算を途中終了し，新たなパラメータを入力して新たな計算を開始することができる．

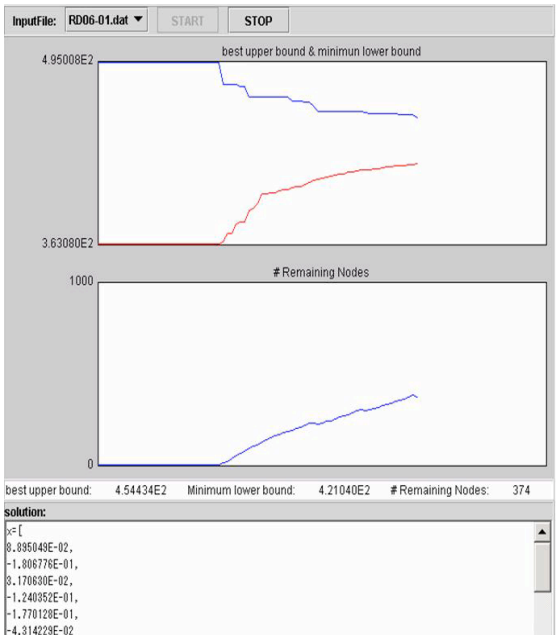

図 9．ユーザ画面

5. 関連研究

分散計算システム上での細粒度アプリケーションに関する研究が文献[17][18]において報告されている．これら研究では，本稿と同様に階層的な手法を用いてアプリケーションを構築しているが，[17]では，専有ネットワークによって接続された PC クラスタ上での細粒度アプリケーションの性能を評価している．本評価では，PC クラスタを接続する専有ネットワーク上の遅延およびバンド幅のエミュレーションを行い，ネットワーク性能がアプリケーションに与える性能を議論しているが，実際のグリッド環境上での評価結果は報告されていない．[18]では，グリッド上で divide-and-conquer アプリケーションを開発するためのプログラミング環境である Satin/Ibis を提案し，40CPU から成るグリッド実験環境上での評価結果を示している．Satin/Ibis は divide-and-conquer アプリケーションに特化したシステムであり，本稿が対象とする分枝限定法の実装に本システムをそのまま用いることはできない．また，本稿のアプリケーションの実装に用いた GridRPC は特定のアプリケーションに特化したものではなく，また現在標準化も進められていることから，本稿が示した実装方法は，グリッド上の様々な細粒度アプリケーションに適用可能である．

6. まとめ

本稿では，グリッド上で並列分枝限定法アプリケーションを効率よく実行するための手法とその性能評価結果について報告した．本アプリケーションをグリッド実験環境上に実装し，性能評価を行った結果，Ninf-G および Ninf を組み合わせて実装された階層的マスタ・ワーカ方式を用いた本アプリケーションは，実行時間が 1[sec]未満の細粒度タスクから構成される並列分枝限定法アプリケーションをグリッド上で効率よく実行可能であることが確認された．

本評価で使用した擬似グリッド実験環境は，現状では通信遅延を擬似的に再現しているのみである．実際の WAN 上の通信はより複雑なふるまいを持つため，著者らは，より洗練されたモデルを用いた擬似グリッド実験環境を構築する予定である．また，PC クラスタ間の負荷分散に関しても，オーバーヘッド等に関するより詳細な評価を進め，アルゴリズムの検討を進める予定である．

謝辞　本研究の一部は，科学技術振興機構計算科学技術活用型特定研究開発推進事業（ACT-JST）研究課題「コモディティグリッド技術によるテラスケール大規模数理最適化」の援助による．また，本研究について日頃よりご助言いただいている Ninf プロジェクトの皆様に感謝いたします．

参　考　文　献

[1] J. Goux, S. Kulkarni, J. Linderoth, and M. Yoder, An enabling framework for master-worker applications on the computational grid, In *Proc. the 9th IEEE Symposium on High Performance Distributed Computing (HPDC9)*, 2000.

[2] E. Heymann, M. A. Senar, E. Luque, and M. Livny, Adaptive scheduling for master-worker applications on the computational grid, *Proc. of the 1st IEEE/ACM International Workshop on Grid Computing (Grid2000)*, 2000.

[3] M. O. Neary and P. Cappello, Advanced Eager Scheduling for Java-Based Adaptively Parallel Computing, *Proc. of the 2002 joint ACM-ISCOPE conference on Java Grande*, 2002.

[4] H. Takemiya, K. Shudo, Y. Tanaka and S. Sekiguchi. Development of Grid Applications on Standard Grid Middleware, *Porc. of the GGF8 Workshop on Grid Applications and Programming Tools*, 2003.

[5] H. Fujioka and K. Hoshijima. Bounds for the bmi eigenvalue problem. Trans. of the Society of Instrument and Control Engineers, 33(7):616–621, 1997.

[6] M. Fukuda and M. Kojima, Branch-and-Cut Algorithms for the Bilinear Matrix Inequality Eigenvalue Problem, *Computational Optimization and Applications*, 19(1):79-105, 2001.

[7] H. Kasahara and S. Narita, Practical Multiprocessor Scheduling Algorithms for Efficient Parallel Processing, *IEEE Trans. on Computers,* C-33(11), pp.1023-1029, 1984.

[8] K. Aida, W. Natsume and Y. Futakata, Distributed Computing with Hierarchical Master-worker Paradigm for Parallel Branch and Bound Algorithm, *Proc. of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2003)*, 2003.

[9] Y. Tanaka, H. Nakada, S. Sekiguchi, T. Suzumura and S. Matsuoka, Ninf-G: A Reference Implementation of RPC-based Programming Middleware for Grid Computing, *J. of Grid Computing*, 1(1):41-51, 2003.

[10] S. Matsuoka, H. Nakada, M. Sato and S. Sekiguchi, Design Issues of Network Enabled Server Systems for the Grid, *Grid Computing – Grid 2000, LNCS1971*, pp.4-17, 2000.

[11] K. Aida, Y Futakata and S, Hara, High-performance Parallel and Distributed Computing for the BMI Eigenvalue Problem, Proc. 16th IEEE International Parallel and Distributed Processing Symposium (IPDPS 2002), 2002

[12] 大角，合田，階層的マスタワーカ方式を用いたグリッドアプリケーションにおける負荷分散の性能評価，情報処理学会 HPC 研究会，2004 年 8 月

[13] Global Grid Forum, *http://www.ggf.org/*

[14] I. Foster and C. Kesselman, Globus: A Metacomputing Infrastructure Toolkit, *Int. J. of Supercomputing Applications*, 11(2):115-128, 1997.

[15] ApGrid, http://www.apgrod.org/

[16] NIST Net, http://snad.ncsl.nist.gov/nistnet/

[17] Plaat, H. E. Bal and R. F. Hofman, Sensitivity of Parallel Applications to Large Differences in Bandwidth and Latency in Two-Layer Interconnects, *Porc. of High Performance Computer Architecture (HPCA-5)*, pp. 244-253, 1999.

[18] R. van Nieuwpoort, J. Massen, T. Kielmann and H. E. Bal, Satin: Simple and Efficient Java-based Grid Programming, Proc. *Workshop on Adaptive Grid Middleware (AGridM 2003)*, 2003.